# Magiczne Bloczki

**Dokumentacja**

Autor
Rafał Baran
Data 3.7.2005

## Spis treści

# 1. Wstęp

Program Magiczne Bloczki umożliwia projektowanie oraz analizę algorytmów. Dzięki prostej obsłudze jest doskonały do nauki i zrozumienia działania podstawowych jak i zaawansowanych algorytmów. Wizualizacja oraz projektowanie algorytmów jest zrealizowana za pomocą schematów blokowych. Wbudowany kompilator umożliwia sprawdzanie poprawności jak i symulowanie algorytmu.

# 2. Środowisko projektowe

Środowisko projektowe zostało podzielone na cztery części:

a) obszar znajdujący się po prawej stronie, na którym możemy rysować algorytm
b) obszar znajdujący się po lewej stronie, zawiera: listę aktualnie projektowanych algorytmów (otwartych plików), nawigator (dzięki czemu możemy w łatwy i szybki sposób przemieszczać się po całym algorytmie) oraz dynamiczną pomoc.
c) obszar znajdujący się w górnej części zawierający menu główne oraz paski narzędzi
d) obszar znajdujący się w dolnej części zawierający pasek informacyjny

## *2.1 Opis podstawowych narzędzi*

### 2.1.1 Menu główne

Menu główne znajduje się w górnej części głównego okna. Menu składa się z następujących elementów:

a) Plik
b) Edycja
c) Uruchom
d) Opcje
e) Widok
f) Pomoc

Ad a) Menu *Plik*

Menu plik składa się z następujących elementów:

| | | |
|---|---|---|
|  | Nowy schemat<br>(Ctrl + N) | Umożliwia tworzenie nowego schematu(algorytmu) |
|  | Otwórz<br>(Ctrl + O) | Otwiera wcześniej zapisany algorytm |
|  | Zapisz<br>(Ctrl + S) | Umożliwia zapisywanie projektowanych algorytmów |
| | Zapisz jako ... | Umożliwia zapisywanie projektowanych algorytmów pod inną nazwą |
| | Eksportuj do schowka | Eksportuje zawartość aktualnie projektowanego algorytmu w postaci grafiki (format wektorowy). Opcja przydatna jeżeli chcesz szybko wstawić zaprojektowany algorytm do edytora tekstu lub na stronę www. Wówczas wystarczy wyeksportować algorytm i wkleić(Ctrl + V) w dowolnym edytorze. |
| | Eksportuj do pliku | Eksportuje zawartość aktualnie projektowanego algorytmu w postaci grafiki (format emf lub jpg) do pliku. |
| | Zamknij | Umożliwia zamknięcie aktualnie projektowanego algorytmu |
| | Ustawienia drukarki | Umożliwia ustawić parametry drukarki |
| | Drukuj | Drukuje aktualnie projektowany algorytm. (W celu profesjonalnego wydruku algorytmu zaleca się skorzystanie z opcji Eksportuj do schowka) |
| | Koniec<br>(Ctrl + Q) | Zamyka program |

Ad b) Menu *Edycja*

Menu plik składa się z następujących elementów:

| | | |
|---|---|---|
| | Cofnij<br>(Ctrl + Z) | Umożliwia cofnięcie ostatniej wykonanej czynności |
|  | Wytnij<br>(Ctrl + X) | Umożliwia wycięcie jednego lub kilku obiektów algorytmu(bloki są wycinane razem z połączeniami). |

| | |
|---|---|
|  Kopiuj (Ctrl + C) | Umożliwia skopiowanie jednego lub kilku bloków algorytmu |
|  Wklej (Ctrl + V) | Umożliwia wklejenie zawartości schowka |
| Usuń połączenia (Ctrl + D) | Usuwa połączenia z innymi obiektami dla aktualnie zaznaczonego obiektu |
|  Przesuń obszar roboczy (Ctrl + 1) | Umożliwia przesuwanie obszaru roboczego |
|  Wybieranie i przesuwanie bloków (Ctrl + 2) | Umożliwia zaznaczanie jednego lub kilku obiektów oraz ich przesuwanie |
|  Blok rozpoczynający program (Ctrl + 3) | Umożliwia wstawianie na obszar roboczy bloku rozpoczynającego algorytm |
|  Blok kończący program (Ctrl + 4) | Umożliwia wstawianie na obszar roboczy bloku kończącego algorytm |
|  Blok przetwarzania danych (Ctrl + 5) | Umożliwia wstawianie na obszar roboczy bloku przetwarzania danych |
|  Blok warunkowy (Ctrl + 6) | Umożliwia wstawianie na obszar roboczy bloku warunkowego |
|  Blok wejścia/wyjścia (Ctrl + 7) | Umożliwia wstawianie na obszar roboczy bloku wejścia/wyjścia |
|  Węzeł pomocniczy (Ctrl + 8) | Umożliwia wstawianie na obszar roboczy węzła pomocniczego. Węzeł pomocniczy umożliwia tworzenie linii łamanych oraz łączenie kilku połączeń w jedno. |
|  Notatka/Opis (Ctrl + 9) | Umożliwia wstawianie na obszar roboczy notatki oraz dodanie lokalnego opisu |
|  Wyśrodkuj w pionie (Ctrl + [) | Umożliwia wyśrodkowanie w pionie kilku obiektów względem siebie. Obiekty muszą być zaznaczone. |
|  Wyśrodkuj w poziomie (Ctrl + ]) | Umożliwia wyśrodkowanie w poziomie kilku obiektów względem siebie. Obiekty muszą być zaznaczone. |

Ad c) Menu *Uruchom*

Menu plik składa się z następujących elementów:

| | |
|---|---|
| Kompiluj algorytm (Ctrl + F9) | Umożliwia kompilację zaprojektowanego algorytmu |
|  Uruchom algorytm (F9) | Umożliwia kompilację i uruchomienie zaprojektowanego algorytmu |
|  Uruchom algorytm krok po kroku (F4) | Umożliwia kompilację i uruchomienie pierwszej instrukcji algorytmu, następnie przejście w tryb krok po kroku. |
|  Wstrzymaj (F12) | Umożliwia chwilowe wstrzymanie wykonywania algorytmu |
|  Wznów wykonywanie (F11) | Umożliwia wznowienie (tylko po wcześniejszym wstrzymaniu) wykonywania algorytmu |
|  Następna instrukcja (F8) | Umożliwia wykonanie pojedynczej instrukcji oraz przejście do następnej instrukcji |
|  Następny blok (F6) | Umożliwia wykonanie całego bloku instrukcji oraz przejście do kolejnego bloku |
|  Następna mikro instrukcja (F7) | Umożliwia wykonanie kolejnej mikroinstrukcji dla aktualnej instrukcji. (Każda instrukcja/wyrażenie jest rozkładane na szereg prostych/niepodzielnych instrukcji np.: sumujących, porównujących) |
|  Zakończ wykonywanie algorytmu (F2) | Umożliwia zakończenie aktualnie wykonywanego algorytmu. |

Ad d) Menu *Opcje*

Menu plik składa się z następujących elementów:

| | |
|---|---|
| Konfiguracja | Umożliwia wywołanie okna z konfiguracją programu(patrz rozdział 2.2.1) |
| Siatka | Umożliwia włączenie lub wyłączenie siatki, do której wyrównywane są obiekty podczas przesuwania. Wybranie rozmiaru siatki umożliwia podmenu, które aktywuje się po wybraniu opcji Siatka. |

Ad e) Menu *Widok*

Menu plik składa się z następujących elementów:

| | |
|---|---|
|  Analiza programu (Ctrl + R) | Umożliwia pokazanie lub ukrycie okna umożliwiającego analizę algorytmu. |
|  Kod źródłowy (Ctrl + K) | Umożliwia pokazanie lub ukrycie okna zawierającego kod skompilowanego algorytmu |
| Pokaż/ukryj węzeł | Umożliwia pokazanie lub ukrycie węzłów pomocniczych |
| Połączenia łamane | Umożliwia włączenie lub wyłączenie połączeń łamanych dla wszystkich obiektów |
| Paski narzędzi | Umożliwia pokazanie lub ukrycie jednego z kilku pasków narzędziowych. |

Ad f) Menu *Pomoc*

Menu plik składa się z następujących elementów:

| | |
|---|---|
| Pomoc (Ctrl + K) | Umożliwia wyświetlenie pomocy dla programu |
| Strona www programu | Umożliwia wywołanie w domyślnej przeglądarce internetowej oficjalnej strony www na temat programu. (Ze strony www można pobierać aktualizacje programu) |
| Rejestracja | Umożliwia zarejestrowanie programu lub przeglądanie danych na temat licencji. (Opcja dostępna w wersji komercyjnej) |
| O programie | Umożliwia wyświetlenie informacji na temat programu |

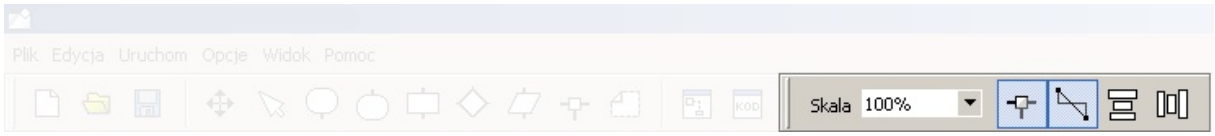
### 2.1.2 Pasek narzędzi - standardowy

Standardowy pasek narzędzi umożliwia wykonywanie podstawowych czynności dotyczących: otwierania i zapisywania algorytmów, tworzenia i zarządzania blokami. Wszystkie przyciski znajdujące się na pasku są odpowiednikami opcji z menu głównego(patrz opis w rozdziale 2.1.1). Domyślnie pasek znajduje się w górnej części głównego okna. Poniżej znajduje się rysunek przedstawiający standardowy pasek narzędzi:


**Rysunek 1 - Pasek narzędzi: Standardowy**

### 2.1.3 Pasek narzędzi – widok

Pasek narzędzi *Widok* umożliwia wykonywanie podstawowych czynności dotyczących: wyglądu obiektów oraz skali w jakiej przedstawiane są obiekty. Wszystkie przyciski znajdujące się na pasku są odpowiednikami opcji z menu głównego(patrz opis w rozdziale 2.1.1). Domyślnie pasek znajduje się w górnej części głównego okna. Poniżej znajduje się rysunek przedstawiający pasek narzędzi dotyczący widoku:




**Rysunek 2 - Pasek narzędzi: Widok**

## *2.2 Opis poszczególnych okien programu*

### 2.2.1 Okno - Konfiguracja

Omawiane w tym rozdziale okno umożliwia konfigurację programu. Poniżej zostały opisane wszystkie opcje udostępniane przez to okno:

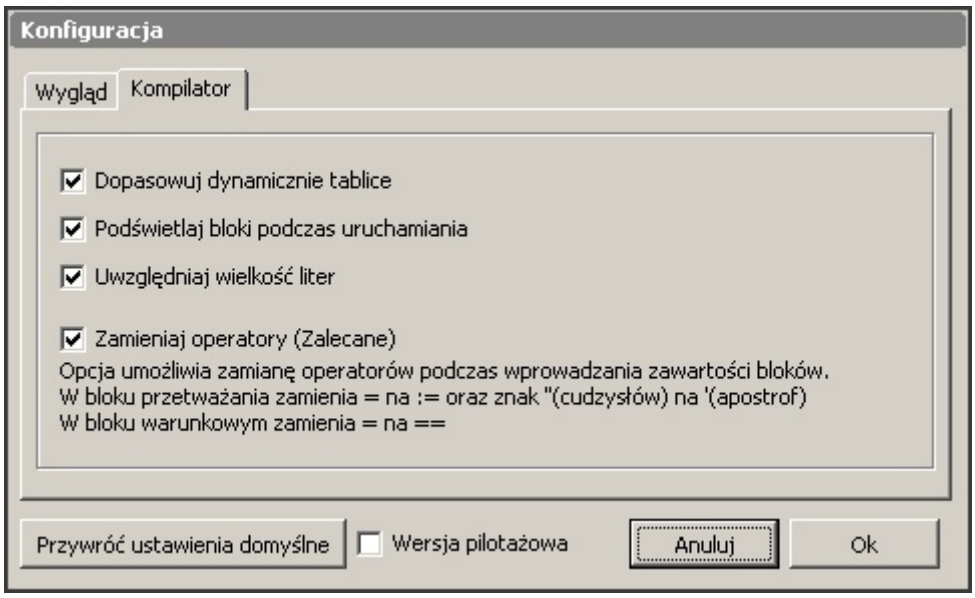
Zakładka *wygląd*:

a) *Wypełnienie* – umożliwia ustawienie domyślnego koloru wypełnienia bloków
b) *Obrys* - umożliwia ustawienie domyślnego koloru obrysu bloków
c) *Czcionka* - umożliwia ustawienie domyślnego kroju i rozmiaru czcionki dla tekstu zawartego w blokach.

Zakładka *kompilator*:

a) *Dopasowuj dynamicznie tablice* – umożliwia włączenie lub wyłączenie automatycznej zmiany wielkości(rozmiaru) tablicy. Jeżeli opcja jest włączona to każde odwołania do tablicy poza jej rozmiar spowoduje automatyczne zwiększenie rozmiaru tablicy, tak aby element do którego się odwołujemy był dostępny.
b) *Podświetlaj bloki podczas uruchamiania* – umożliwia włączenie lub wyłączenie podświetlania(w kolorze zielonym) bloków w trakcie analizy algorytmu. Podświetlenie bloku informuje nas o tym że aktualnie wykonywane są instrukcje zawarte w tym bloku.
c) *Uwzględniaj wielkość liter* – umożliwia włączenie lub wyłączenie reagowania na duże i małe litery w nazewnictwie zmiennych i tablic. Jeżeli opcja jest włączona kompilator będzie rozróżniał zamienną o nazwie *Tablica* od zmiennej o nazwie *tablica*, ponieważ różnią się one wielkością pierwszej litery.
d) *Zamieniaj Operatory* – umożliwia włączenie lub wyłączenie automatycznej podmiany operatorów, które mogą być traktowane w sposób niejednoznaczny. Spowodowane jest to faktem, że w programie zostały wykorzystane operatory zaczerpnięte z różnych języków programowania. Zlecane jest włączenie tej opcji.

Poniżej zostało przedstawione okno konfiguracji programu:




## 2.2.2 Okno – Analiza programu

Omawiane w tym rozdziale okno umożliwia sterowanie analizą programu. W górnej części okna znajdują się przyciski które umożliwiają kolejno:

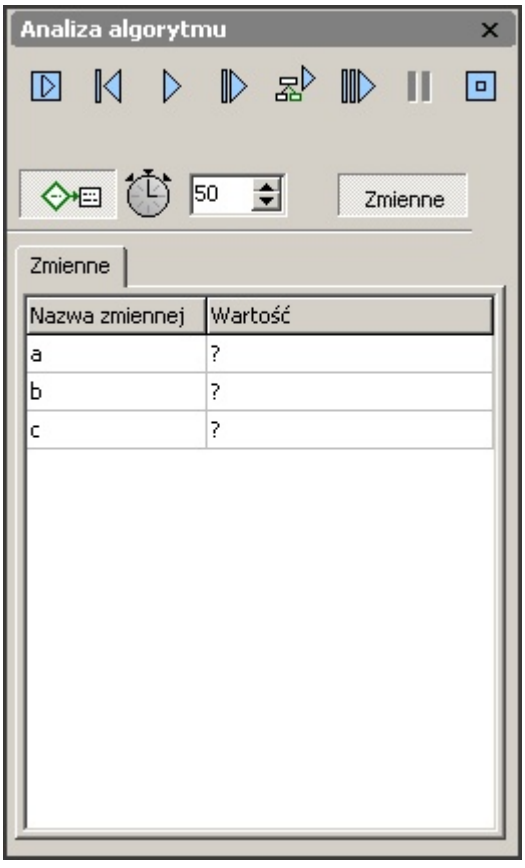
- Uruchomienie algorytmu
- Uruchomienie algorytmu krok po kroku (Przejście w tryb analizy)

- Wznowienie wykonywania algorytmu
- Przejście do następnej instrukcji i wykonanie jej
- Przejście do następnej mikro instrukcji i wykonanie jej
- Przejście do następnego bloku i wykonanie instrukcji w nim zawartych
- Wstrzymanie wykonywania algorytmu
- Zakończenie wykonywania algorytmu

Tuż pod przyciskami sterowania znajduje się opcja umożliwiająca włączenie lub wyłącznie podświetlania bloków(analogiczna do tej znajdującej się w oknie konfiguracji programu – patrz rozdział 2.2.1) oraz definiowanie z jakim opóźnieniem mają podświetlać się kolejne bloki podczas działania algorytmu.

W dolnej części okna znajduje się lista wszystkich zmiennych(wraz z wartościami jakie przechowują) używanych w algorytmie. Lista zmiennych dostępna jest tylko w trybie analizy algorytmu (patrz rozdział 5).

Poniżej znajduje się rysunek przedstawiający okno analiza programu:



### 2.2.3 Okno – Kod źródłowy

Omawiane w tym rozdziale okno umożliwia różnego rodzaju pomocnicze funkcje. Wszystkie z nich zostały przedstawione poniżej według zakładek:

a) Algorytm
b) Drzewo
c) Punkty przerwania
d) Konsola

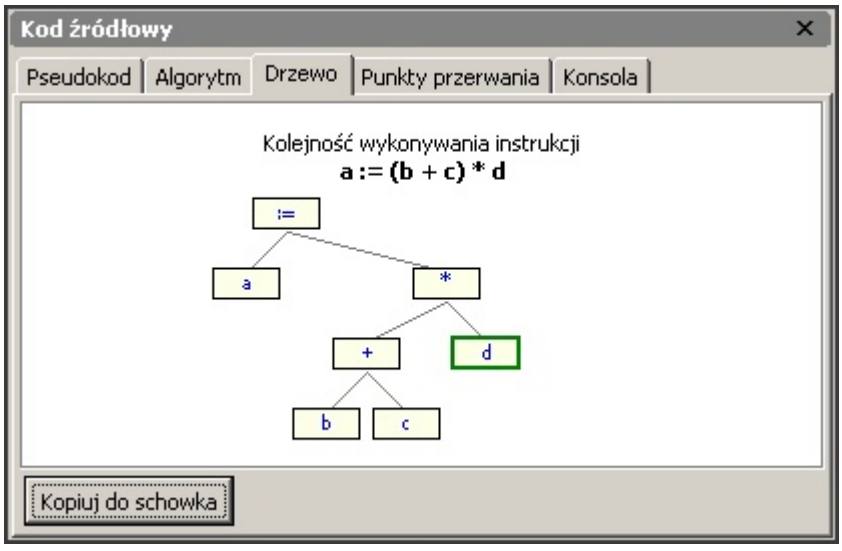
*ad b) Algorytm*

Zakładka ta pokazuje wszystkie kolejne instrukcje zaprojektowanego algorytmu w postaci opisu. Bieżąca instrukcja jest zawsze podświetlona na zielono, natomiast kolejna na szaro. Umożliwia to łatwiejsze zrozumienie algorytmu.

*ad c) Drzewo*

Zakładka zawiera graficzną reprezentację bieżącej instrukcji. Wizualizacja przedstawiona jest w postaci drzewa. Symbolizuje ono kolejność wykonywania instrukcji cząstkowych(mikroinstrukcji) z których składa się bieżąca instrukcja. Mechanizm ten umożliwia zrozumienie kolejności wykonywania operatorów. Dzięki funkcji *Następna mikro instrukcj*(patrz rozdział 2.1) możliwe jest śledzenie kolejnych instrukcji cząstkowych oraz wartości pośrednich. Bieżąca mikroinstrukcja została oznaczona pogrubioną zieloną ramką. Każde drzewo można w łatwy sposób przenieść do dowolnego edytora tekstu lub programu graficznego za pomocą opcji *Kopiuj do schowka*.

Wygląd zakładki został przedstawiony poniżej:

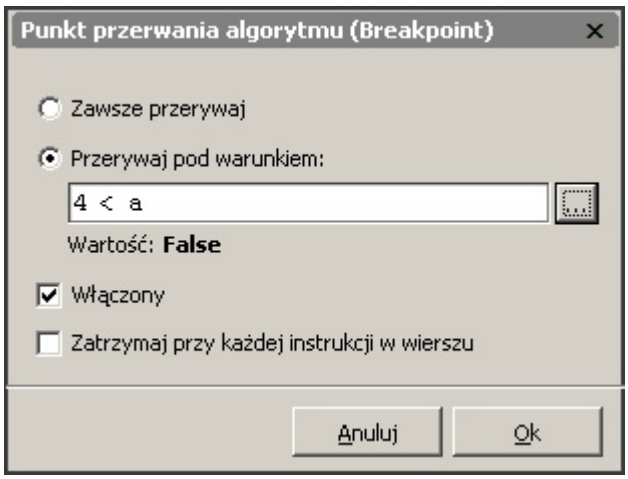


*ad d) Punkty przerwania*

Punkty przerwania algorytmu umożliwiają definiowanie miejsc, w których ma zostać wstrzymane wykonywanie algorytmu podczas jego wykonywania. Dokładniej mówiąc punkty przerwania pozwalają na szybką i skuteczną analizę algorytmu. Można je stosować w takich przypadkach jak:

- chcemy przeanalizować końcową część algorytmu, ale pierwsza część wykonuje się zbyt długo żeby można było wykonywać ją krok po kroku. W tym celu zakładamy w interesującym nas miejscu punkt przerwania i uruchamiamy algorytm w normalnym trybie (opcja *Uruchom->Uruchom algorytm F9*)
- chcemy zacząć analizę zaprojektowanej pętli od pewnego n-tej iteracji. W tym celu zakładamy w pętli warunkowy punkt przerwania. Następnie definiujemy warunek przerwania np.: `4 < a` i uruchamiamy algorytm w normalnym trybie. Unikniemy w ten sposób żmudnego przechodzenia krok po kroku do kolejnej iteracji pętli.

Poniżej zostało przedstawione okno za pomocą którego możemy definiować punkty przerwania:

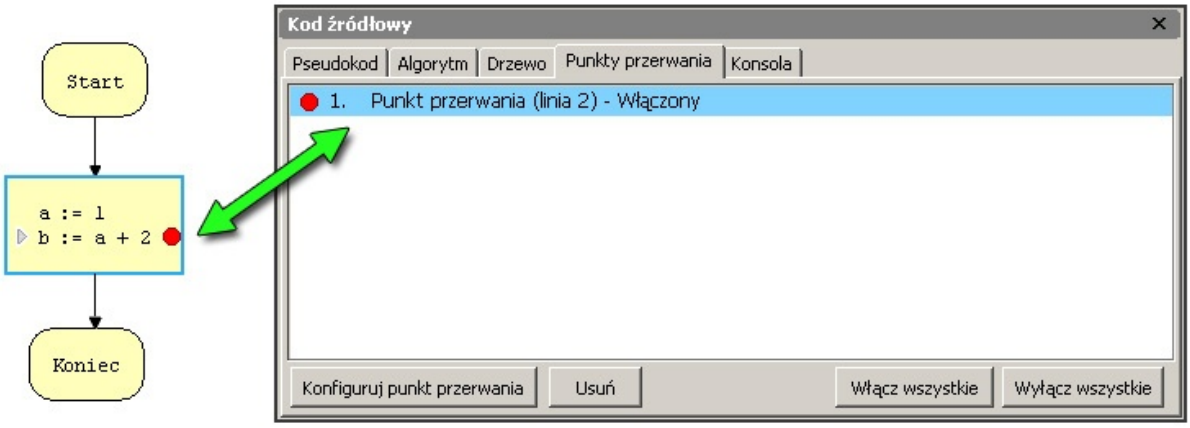



Przycisk oznaczony trzema kropkami(...) umożliwia obliczenie i sprawdzenie czy równanie lub nierówność zwraca prawdę(True) czy fałsz(False).

Opcja *wyłączony* umożliwia włączenie lub tymczasowe wyłączenie punktu przerwania.
*Opcja Zatrzymaj przy każdej instrukcji w wierszu* umożliwia tak jak sama nazwa wskazuje wstrzymywanie działania algorytmu przy każdej instrukcji w wierszu. Jest to pomocne w momencie kiedy w jednym wierszu wpiszemy kilka instrukcji oddzielając każdą z nich średnikiem.

Punktu przerwania dodajemy poprzez naciśnięcie lewym przyciskiem myszki na prawy margines bloczka. Czynność ta spowoduje automatycznie dodanie punktu przerwania algorytmu na listę wszystkich punktów przerwania znajdującej się na omawianej zakładce.

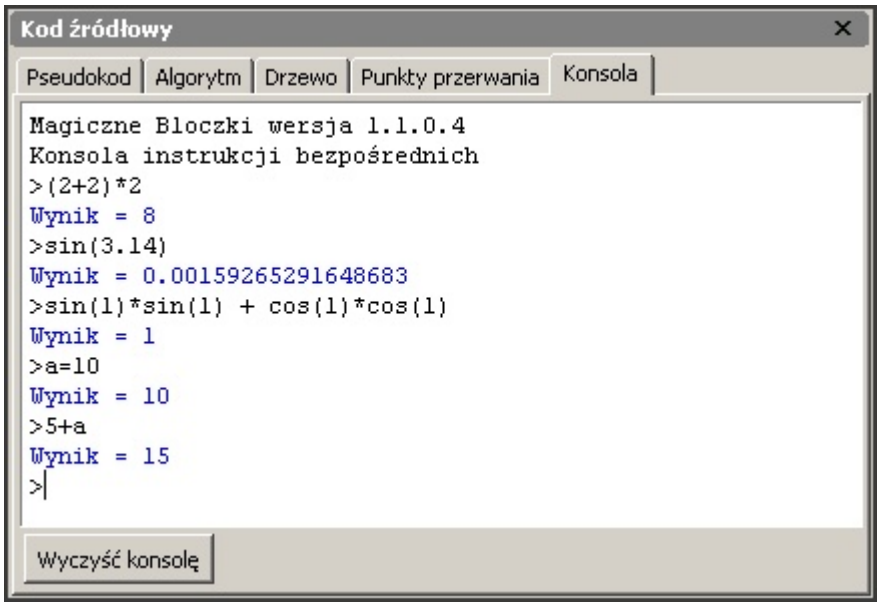
Przedstawia to poniższy rysunek:




*ad e) Konsola*

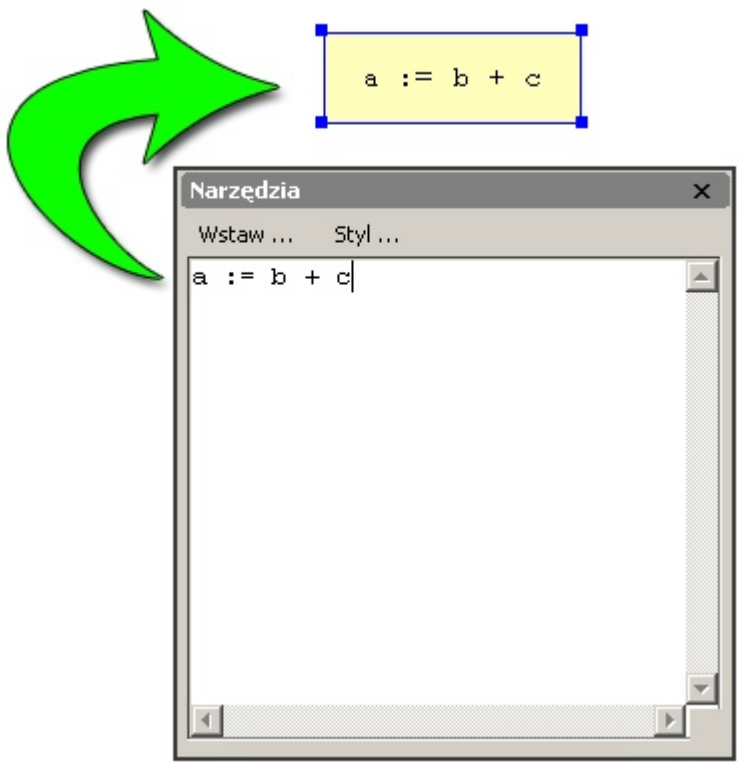
Zakładka ta jak sama nazwa wskazuje udostępnia konsolę, dzięki której możemy dokonywać szybkich obliczeń. W tym celu możemy także wykorzystywać zmienne zdefiniowane w naszym algorytmie. Z poziomu konsoli możemy również przypisywać zmiennym nowe wartości.

Poniżej znajduje się rysunek przedstawiający konsolę:



### 2.2.4 Okno – Narzędzia

Omawiane w tym rozdziale okno umożliwia wprowadzanie kodu źródłowego dla poszczególnych bloków. Okno pojawia się w momencie zaznaczenia bloku przeważania danych, bloku wejścia/wyjścia lub bloku warunkowego. Co zostało przedstawione na poniższym rysunku:




Dodatkowo okno udostępnia dwie pomocne opcje:

a) Wstaw... - umożliwia wstawianie Słów kluczowych, operatorów i funkcji.
b) Styl ... – umożliwia zmianę koloru oraz czcionki dla zaznaczonego bloku.

# 3. Opis wbudowanego języka programowania

Poniższy rozdział przedstawia jak poprawnie napisać kod źródłowy dla każdego bloczka. Jest to ważne ponieważ wbudowany język programowania umożliwia analizowanie tylko bezbłędnie zaprojektowanego algorytmu. Zapoznaj się z tym rozdziałem jeszcze przed projektowaniem algorytmu. Jeżeli kiedykolwiek miałeś styczność z jakimś językiem programowania i nie są Ci obce takie pojęcia jak *Zmienne*, *Operatory*, itp. skup raczej uwagę na składni języka i przeanalizuj dostępne operatory i funkcje niż wnikaj w dokładny opis pojęcia *Zmienna*, itp.

### 3.1 Zmienne i tablice

Podstawą funkcjonowania każdego algorytmu są tzw. Zmienne, które umożliwiają przechowywanie pewnych wartości obliczanych w trakcie wykonywania algorytmu. Można sobie wyobrazić taką sytuację: Za pomocą kalkulatora chcemy obliczyć pierwiastki równania kwadratowego $x^2 + 3x + 2 = 0$. Pierwszą czynnością jaką musimy wykonać jest obliczenie delty dla tego równania ($\Delta = b^2-4ac$ *przy. autor*) uzyskany wynik musimy więc gdzieś zapamiętać, ponieważ posłużymy się nim do kolejnych obliczeń. Właśnie tutaj wykorzystamy zmienne. Załóżmy że mamy zmienną o nazwie delta, do której przypiszemy naszą wyliczoną wartość. Zwróć uwagę na słowo „przypiszemy" ma ono bardzo ważne znaczenie, które postaram się opisać na poniższym przykładzie dotyczącym naszych obliczeń:

Delta = b*b – 4*a*c (dla uproszczenia dodam że w przykładzie użyliśmy $b^2 \leftrightarrow$ b*b, na razie nie zwracaj uwagi na zapis b*b – 4*a*c, zrozumiesz to czytając kolejny rodział 3.2 dotyczący operatorów)

Przedstawiony powyżej zapis przedstawia typową operację przypisania wartości do zmiennej. Z lewej strony znaku równości znajduje się nazwa zmiennej, pod którą chcemy przypisać wartość znajdującą się po prawej stronie znaku równości(co prawda naszym przypadku zamiast wartości mamy pewne wyrażenie matematyczne, ale należy to rozumieć w taki sposób że na podstawie tego wyrażenia zostanie wyliczona pewna wartość).

Wracając do naszego równania spróbujemy wykorzystać naszą zmienną do dalszych obliczeń. Jak już wspomniałem wcześniej zmiennym możemy przypisywać wartości, możemy także jak wskazuje na to prosta logika odczytywać te wartości np.: używając zmiennej(a raczej jej wartości) do obliczenia kolejnego wyrażenia. Przykład takiej czynności znajduje się poniżej:

x = delta + 1 (podany wzór niema nic wspólnego z obliczeniem pierwiastków równania kwadratowego, ma na celu jedynie przedstawić w prosty sposób jak można wykorzystywać zmienne).

Warto w tym momencie ustalić jeszcze jedną rzecz, w omawianym programie operatorem przypisania jest symbol := (dwukropek następnie znak równości) i tylko takiego operatora można używać w celu przypisywania wartości zmiennym. Przedstawiony w powyższym przykładzie operator przypisania = (równa się) ma tylko ułatwić zrozumienie tematu, ponieważ dokładny opis operatorów znajduje się dopiero w następnym rozdziale.

Myślę że na tym krótkim wstępie poprzestaniemy(obliczenie pierwiastków równania kwadratowego nie jest celem tego rozdziału) i przejdziemy do tematu jakim jest deklaracja zmiennych.

W większości języków programowania jeżeli chcesz użyć jakiejś zmiennej musisz ją zadeklarować, trochę inaczej jest w omawianym programie, ponieważ dokonuje on analizy napisanego przez Ciebie kodu źródłowego i automatycznie wykrywa, w których miejscach została użyta zmienna. Jest to spore ułatwienie ponieważ skupiasz więcej uwagi na myśleniu nad algorytmem. Pragnę jednak powiedzieć, że dobrym zwyczajem jest po zakończeniu(o ile nie na początku) projektowania algorytm zdefiniować na początku programu lub przynajmniej w blokach opisowych wszystkie użyte przez Ciebie zmienne.

Służy do tego słowo kluczowe:
**Dim** *nazwaZmiennej1*, *[nazwaZmiennej2], ...*

Gdzie:
*nazwaZmiennej1* – to nazwa zmiennej, którą chcemy zadeklarować
Dopuszczalne jest deklarowanie kilku zmiennych oddzielając ich nazwy przecinkiem

Przykład 1:

```
Dim delta
Dim x1, x2
....
```

W omawianym programie podczas deklaracji zmiennych nie musisz podawać typów zmiennych (tzn. nie musisz określać czy będą one przechowywały tekst, czy też liczby) tak jak ma to miejsce w innych językach programowania.

Kolejną istotną rzeczą jak zostanie omówiona są tablice. Tablice są to pewnego rodzaju zmienne, które potrafią przechowywać więcej niż jedną wartość. Obecnie większość algorytmów wykorzystuje tablice, dlatego postaram się przybliżyć do jakich celów mogą one posłużyć. Wyobraź sobie sytuację, w  której musisz przechować pewien zbiór 3 liczb(nazwa zbiór nie została użyta przypadkowo w większości literatury poświęconej algorytmom tablice są kojarzone ze zbiorami matematycznymi), załóżmy że liczby te wprowadzasz z klawiatury. Następnie chcesz żeby program wyświetlił je w kolejności od najmniejszej do największej.
Jeżeli chwilę się zastanowisz dojdziesz do wniosku że wszystkie wpisywane z klawiatury liczby należy gdzieś zapamiętać, a na końcu porównać każdą liczbę z każdą i wyświetlić w odpowiedniej kolejności. Właśnie do tego celu można wykorzystać tablicę. Poniżej zostało opisane jak deklaruje się tablicę oraz został przedstawiony przykład deklaracji tablicy i przypisania każdemu elementowi tej tablicy pewnej wartości.

Deklaracja tablicy:
**Dim** *nazwaTablicy[ileElementow], ...*

Gdzie:
*nazwaTablicy* – to nazwa tablicy, którą chcemy zadeklarować
Dopuszczalne jest deklarowanie kilku tablic oddzielając ich nazwy przecinkiem oraz deklarację zmiennych razem z deklaracją tablic w jednej linii(przy użyciu jednego słowa kluczowego *Dim*).

Przykład 2 - deklaracji tablicy składającej się z 3 elementów:

```
Dim tablica[3]
Tablica[1] := 45
Tablica[2] := 3
Tablica[3] := 23
....
```

Należy pamiętać, że deklaracja tablic też nie jest wymagana do poprawnego działania algorytmu(rozmiar tablicy zostaje automatycznie zwiększony jeżeli następuje próba odwołania do elementu tablicy, który znajduje się poza rozmiarem tablicy) ale tak jak zostało to powiedziane wcześniej jest dobrym zwyczajem umieścić wszystkie deklaracje zmiennych i tablic w projektowanym algorytmie.

Według powyższych ustaleń prawidłowy jest również zapis:

```
Tablica[1] = 45
Tablica[2] = 3
Tablica[3] = 23
....
```

Program umożliwia wyłączenie automatycznego zwiększania rozmiaru tablicy. Jest to bardzo pomocne w chwili kiedy indeks elementu tablicy wykracza znacznie poza jej rozmiar(np.: poprzez błędnie zaprojektowany algorytm). Umożliwia to opcja znajdująca się w konfiguracji programu(patrz rozdział 2.2.1).

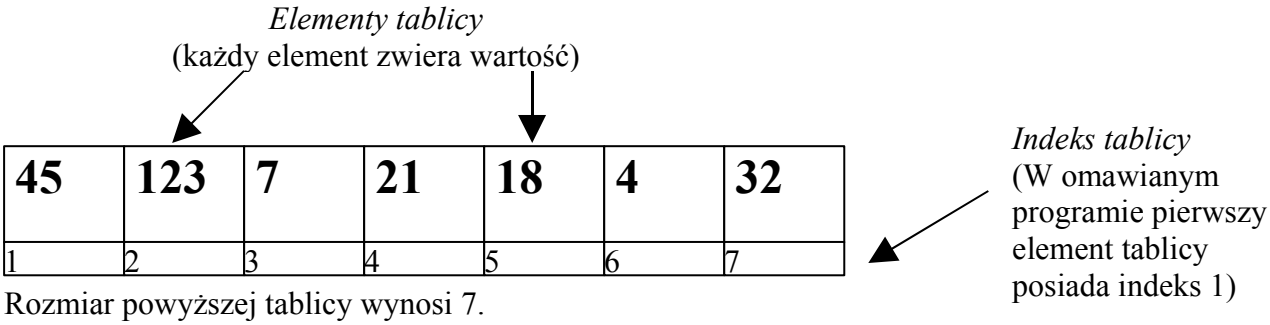
Nadszedł czas na wyjaśnienie jeszcze kilku ważnych pojęć, które są związane z tablicami:

*Rozmiar tablicy* – Określa z ilu elementów(wartości) składa się tablica.

*Element tablicy* – element tablicy jest to wartość, która jest przechowywana pod wskazanym indeksem tablicy

*Indeks tablicy* – jest to kolejny numer licząc od początku tablicy do końca tablicy(rozmiaru tablicy)

Poniżej znajduje się rysunek obrazujący 7 elementową tablicę:




Rozmiar powyższej tablicy wynosi 7.

Wykonując następującą operację przypisania:
Tablica[3] = Tablica[5] + Tablica[7] spowodujemy przypisanie do elementu tablicy o indeksie 3 sumy elementu o indeksie 5 i elementu o indeksie 7, w konsekwencji Tablica[3]=50

Powyższy przykład obrazuje tablice zawierającą 7 liczb, w omawianym programie w tablicy zamiast liczb można przechowywać dowolne wartości tzn.: liczby, tekst, itp. Warto pamiętać również, że każdy element tablicy może przechowywać inne rodzaje wartości (np.: połowa tablicy przechowuje liczby, natomiast druga połowa przechowuje tekst).

Omawiany program umożliwia także deklarowanie tablic wielo wymiarowych, można tego dokonać analogicznie tak jak w przypadku tablic jednowymiarowych dodając tylko kolejną wartość indeksu. Poniżej został przedstawiony przykład deklaracji i użycia tablic wielo wymiarowych:

Przykład 3

```
Dim dwaWymiary[5,6], trzyWymiary[4,5,6]
Dim czteryWymiary[10,3,5,2], itd.

dwaWymiary[1,1] := 1
trzyWymiary[2,5,1] := 2005
czteryWymiary[1,3,5,1] := 6
```

Kolejną rzeczą jaką pragnę omówić to przypisywanie wartości dla całej tablicy. We wcześniejszej części niniejszego rozdziału został omówiony sposób przypisywania wartości do zmiennych niestety w przypadku tablic występuje pewna różnica.

W celu przypisania wartości do jednowymiarowej tablicy należy użyć słowa kluczowego:
**Set** *nazwaTablicy:=(wartosc1, wartosc2, ...)*

Gdzie:
*nazwaTablicy* – to nazwa tablicy do której chcemy wpisać wartości.
W nawiasie należy podać wszystkie elementy(wartości) jakie chcemy wstawić do tablicy.

Przykład 4:

```
Set Tablica:=(5,6,2,4,7)
```

Za pomocą słowa kluczowego Set możemy przypisać wartości także do tablic wielowymiarowych. Przykład został przedstawiony poniżej:

```
Set Tablica:=((5,6,3),(2,4,7))
```

Przykład ten przedstawia przypisanie do tablicy dwu wymiarowej. Nawiasie zostały podane dwa zbiory(dwie tablice elementów) 3 elementowe. Kierując się tą zasadą można wprowadzić dane dla n-wymiarowej tablicy. Należy tylko pamiętać że zamiast wartości możemy wprowadzić cały zbiór.

Według powyższego stwierdzenia możliwy jest także zapis(oparty na przykładzie 4, z tą tylko różnicą, że zamiast piątego elementu został wprowadzony zbiór 3 elementowy)

```
Set Tablica:=(5,6,2,4,(12,42,13))
```

### 3.2 Operatory

Omawiany program udostępnia kilkadziesiąt różnego rodzaju operatorów, dostępnych również w innych językach programowania. Wszystkie operatory można podzielić na kilka grup:

a) operatory przypisania
b) operatory relacji (porównania)
c) operatory arytmetyczne
d) operatory logiczne
e) operacje na bitach

*Ad a) operatory przypisania*

Operatory przypisania służ do przypisywania wartości zmiennym. Przykładowo jeżeli chcemy przechować wartość 5 w zmiennej a, wówczas musimy użyć operatora przypisania:

```
a := 5
```

W omawianym programie najczęściej wykorzystywanym operatorem przypisania jest operator: := (dwukropek i znak równa się). W innych językach programowania(np.:c++, java) stosuję się także operator = (znak równa się). Podczas pisania kodu źródłowego można wprowadzić sam symbol = (znak równa się), ale program automatycznie podmieni go na operator := (dwukropek i znak równa się). Jest to związane z faktem, że początkujący

programiści mylą często operator przypisania = z operatorem relacji ==(dwa znaki równa się). Opcję automatycznego podmieniania można oczywiście wyłączyć(patrz rozdział 2.2.1).

Nie można stosować dwóch i więcej operatorów przypisania w jednej instrukcji(dotyczy to tylko omawianego programu).

Poniżej znajduje się opis podstawowego operatora przypisania(Wszystkie operatory przypisania można znaleźć w dodatku B):

| Operator | Opis | Przykład |
|---|---|---|
| := | Powoduje przypisanie wartości znajdującej się z prawej strony operatora do zmiennej lub tablicy, której nazwa znajdującej się po lewej stronie operatora. | `b := 5`<br>`h[5] := 0`<br>`a := 9*b+4` |

*Ad b) operatory relacji (porównania)*

Operatory relacji służą do porównywania dwóch argumentów(zmiennych lub wartości). Można stosować kilka operatorów relacji w jednej instrukcji(Należy tylko pamiętać o ich priorytetach czyli kolejności wykonania – patrz dodatek B). Poniżej zostaną przedstawione najważniejsze operatory relacji:

| **Operator** | **Opis** | **Przykład** |
|---|---|---|
| == | Znak równości, porównuje dwa argumenty, jeżeli wartość argumentów jest równa wyrażenie zwraca wartość True (prawda) w przeciwnym przypadku zwraca wartość False(fałsz) | `b==5`<br>`4==0`<br>`5+9==9*b` |
| < | Znak mniejszości, porównuje dwa argumenty, jeżeli wartość argumentu z lewej strony operatora jest mniejsza od wartości argumentu z prawej strony wyrażenie zwraca wartość True(prawda) w przeciwnym przypadku zwraca wartość False(fałsz) | `b<5`<br>`6<4`<br>`a<b`<br>`5+9 < 9*b` |
| > | Znak większości, porównuje dwa argumenty, jeżeli wartość argumentu z lewej strony operatora jest większa od wartości argumentu z prawej strony wyrażenie zwraca wartość True(prawda) w przeciwnym przypadku zwraca wartość False(fałsz) | `b>5`<br>`6>4`<br>`a>b`<br>`5+9 > 9*b` |
| <= | Operator mniejsze lub równe , porównuje dwa argumenty, jeżeli wartość argumentu z lewej strony operatora jest mniejsza | `b<=5` |

| | | |
|---|---|---|
| | lub równa od wartości argumentu z prawej strony wyrażenie zwraca wartość True(prawda) w przeciwnym przypadku zwraca wartość False(fałsz) | `6<=4`<br>`a<=b`<br>`5+9 <= 9*b` |
| `>=` | Operator większe lub równe , porównuje dwa argumenty, jeżeli wartość argumentu z lewej strony operatora jest większa lub równa od wartości argumentu z prawej strony wyrażenie zwraca wartość True(prawda) w przeciwnym przypadku zwraca wartość False(fałsz) | `b>=5`<br>`6>=4`<br>`a>=b`<br>`5+9 >= 9*b` |
| `!=` | Znak nierówności, porównuje dwa argumenty, jeżeli wartość argumentu z lewej strony operatora jest różna od wartości argumentu z prawej strony wyrażenie zwraca wartość True(prawda) w przeciwnym przypadku zwraca wartość False(fałsz) | `b!=5`<br>`6!=4`<br>`a!=b`<br>`5+9 != 9*b` |
| `<>` | j.w. - Operator oznacza dokładnie to samo co operator `!=` | `j.w.` |

*Ad c) operatory arytmetyczne*

| Operator | Opis | Przykład |
|---|---|---|
| `+` | Znak dodawania, pozwala dodać dwa argumenty | `c = a + b` |
| `-` | Znak odejmowania, pozwala odjąć dwa argumenty | `c = a - b` |
| `*` | Znak mnożenia, pozwala pomnożyć dwa argumenty | `c = a * b` |
| `/` | Znak dzielenia, pozwala podzielić dwa argumenty. Wynik oraz argumenty są traktowane jako liczby rzeczywiste | `c = a / b`<br>`c = 5 / 2`<br>`Wynik: c=2.5` |
| `div` | Operator dzielenia całkowitego, pozwala podzielić dwa argumenty. Wynik oraz argumenty są traktowane jako liczby całkowite | `c = a div b`<br>`c = 5 div 2`<br>`Wynik: c=2` |
| `%` | Operator modulo, pozwala uzyskać resztę z dzielenie dwóch argumentów. Wynik oraz argumenty są traktowane jako liczby całkowite | `c = a % b` |
| `mod` | j.w | `c = a mod b` |

*Ad d) operatory logiczne*

| Operator | Opis | Przykład |
|---|---|---|
| `and` | Operator wykonuje operację iloczynu logicznego na dwóch argumentach | `c = a and b` |
| `&&` | j.w. | `c = a && b` |
| `or` | Operator wykonuje operację sumy logicznej na dwóch argumentach | `c = a or b` |
| `\|\|` | j.w. | `c = a \|\| b` |
| `not` | Operator wykonuje operację zaprzeczenia logicznej wartości argumentu | `c = not a` |
| | | |

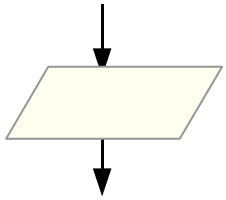
*Ad e) operacje na bitach*

| Operator | Opis | Przykład |
|---|---|---|
| `&` | Operator wykonuje operację iloczynu binarnego na dwóch argumentach | `c = a & b` |
| `\|` | Operator wykonuje operację sumy binarnej na dwóch argumentach | `c = a \| b` |
| `~` | Operator wykonuje operację zaprzeczenia binarnego argumentu | `c = ~a` |
| `shl` | Operator wykonuje przesunięcie bitów opisujących wartość argumentu w lewo | `c = a shl b` |
| `<<` | j.w. | `c = a << b` |
| `shr` | Operator wykonuje przesunięcie bitów opisujących wartość argumentu w prawo | `c = a shr b` |
| `>>` | j.w. | `c = a >> b` |

### 3.3 Operacje wejścia/wyjścia

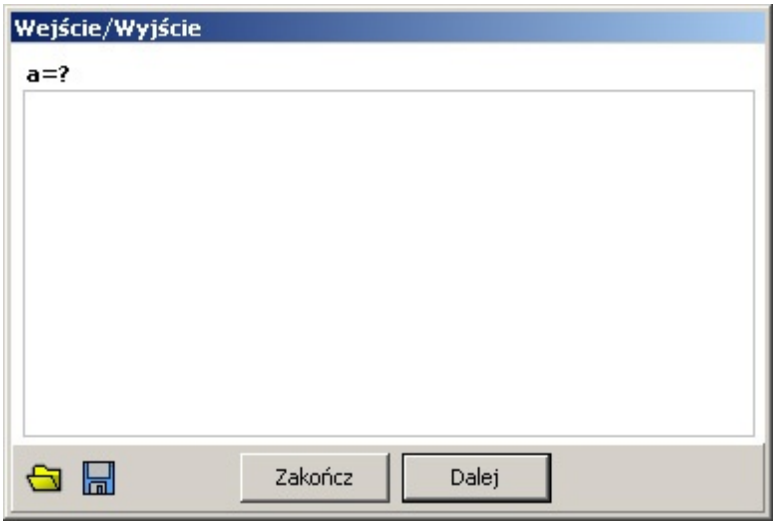
Najważniejszą funkcją oprócz głównej idei każdego algorytmu są przetwarzane dane. Prawie każdy projektowany algorytm zawiera część odpowiedzialną za wprowadzanie danych oraz prezentowanie przetworzonych danych. Do zrealizowania tych czynności wykorzystuje się tak zwane operacje wejścia i wyjścia.

Poniżej został przedstawiony symbol przedstawiający operacje wejścia/wyjścia:



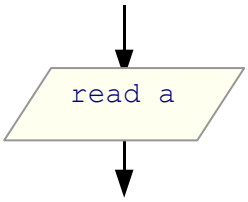
W obmawianym programie za pomocą przedstawionego powyżej symbolu można wprowadzać lub wyświetlać wartości zmiennych oraz tablic w trakcie działania algorytmu.

Do komunikacji z użytkownikiem czyli do wprowadzania i wyświetlania danych służy okno przedstawione poniżej:

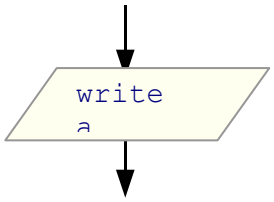



Ze względu na fakt, że omawiany symbol umożliwia zarówno wprowadzanie jak i wyświetlanie danych. Należy określić która z tych czynności ma zostać realizowana. Jeżeli chcemy wprowadzić dane w tym celu musimy użyć polecenie *read*(czytaj przyp. autor) , natomiast jeżeli chcemy wyświetlić dane musimy użyć polecenie *write*(zapisz przyp. autor).

Poniżej znajdują się przykłady z wykorzystaniem obu poleceń:




Powyższy przykład wyświetli okno wejścia/wyjścia i umożliwi wprowadzenie wartości dla zmiennej o nazwie ***a***

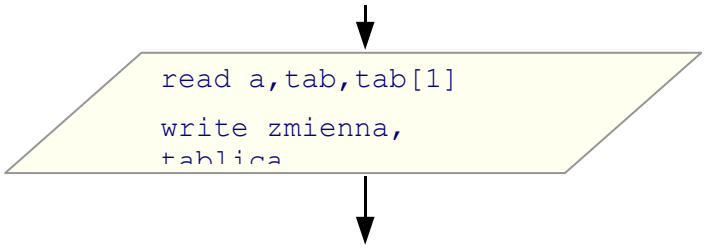



Powyższy przykład wyświetli okno wejścia/wyjścia i wyświetli wartości zmiennej o nazwie ***a***

W celu dokładniejszego opisania poleceń *read* i *write* poniżej przedstawię ich definicję:

```
read nazwaZmiennej,[nazwaTablicy], itd.
write nazwaZmiennej,[nazwaTablicy], itd.
```

Polecenia *read* i *write* można używać w jednym bloku wejścia/wyjścia co zostało przedstawione poniżej:




### 3.4 Funkcje

Omawiany program udostępnia różnego rodzaju wbudowane funkcje. Poniżej znajduje się przykład korzystania z przykładowej funkcji:

```
pierwiastek := sqrt(9)
```

Wszystkie dostępne funkcje zostały opisane w dodatku C.

# 4. Projektowanie algorytmów

W rozdziale tym zostanie przedstawione w jaki sposób poprawnie od podstaw zaprojektować algorytm. Należy dodać że informacje przedstawione poniżej nie dotyczą ogólnej idei tworzenia algorytmów lecz przedstawiają jak można budować lub projektować algorytmy(najczęściej już istniejące) w omawianym programie. Podczas opisu będziemy próbowali stworzyć bardzo prosty algorytm obliczający pole kwadratu.

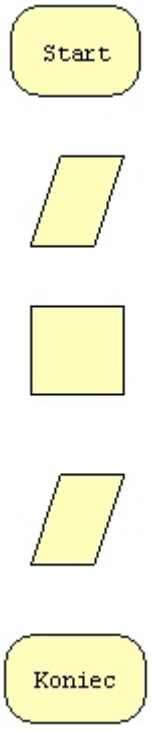
## *4.1 Schemat blokowy algorytmu*

Pierwszym krokiem, który trzeba wykonać to oczywiście stworzenie nowego schematu(opcja *Plik -> Nowy schemat*) a następnie zapisanie go(opcja *Plik -> Zapisz jako ...*) na dysk pod wskazaną dowolną nazwą. Dysponując stworzonym w taki sposób obszarem roboczym możemy dodać pierwsze elementy algorytmu czyli bloki Start i Koniec. W tym celu wybieramy narzędzie  (Blok *Start* – dostępne na pasku standardowym lub poprzez skrót klawiszowy Ctrl + 3) następnie naciskamy lewy przycisk myszy na obszarze roboczym wstawiając tym sposobem blok do naszego algorytmu .
Powyższą czynność powtarzamy wybierając tym razem narzędzie  (Blok *Koniec* - dostępny na pasku standardowym lub poprzez skrót klawiszowy Ctrl + 4).
Kolejny krok polega na wstawieniu dwóch bloków wejścia/wyjścia reprezentowanych poprzez narzędzie  (Skrót klawiszowy Ctrl + 7) oraz jednego bloku przetwarzania danych  (Skrót klawiszowy Ctrl + 5). Wstawiając wymienione elementy postarajmy się o ustawienie ich w odpowiedniej kolejności a mianowicie wymieniając od góry:
Blok start, Blok wejścia/wyjścia, Blok przetwarzania danych, Blok wejścia/wyjścia, Blok koniec. Jeżeli wykonałeś powyższe instrukcje Twój algorytm powinien wyglądać tak jak to zostało przedstawione na rysunku 4.1a.

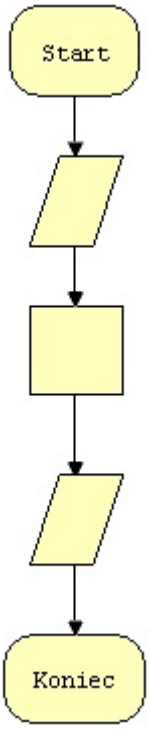
Kolejny krok to odpowiednie połączenie ze sobą bloków. Pamiętaj że połączenia realizowane są za pomocą linii zakończonej strzałką symbolizującą kolejność wykonywania poszczególnych bloków algorytmu. W celu połączenia dwóch bloków ze sobą należy w pierwszej kolejności zaznaczyć(Lewym przyciskiem myszy) blok źródłowy następnie nacisnąć i przytrzymać klawisz Ctrl i wybrać(nacisnąć lewym przyciskiem myszy) blok docelowy. Zabronione są połączenia obustronne bloków np.: blok start z blokiem wejścia/wyjścia i na odwrót. Warto na tym etapie wspomnieć o możliwości usuwania połączeń. Wystarczy zaznaczyć blok z którego chcemy usunąć połączenia i wybrać z menu głównego opcję *Edycja -> Usuń połączenia*(Skrót klawiszowy Ctrl + D). Jeżeli poprawnie połączyłeś Twój algorytm powinien wyglądać tak jak to zostało przedstawione na rysunku 4.1b.

Ostatnim etapem jaki należy wykonać jest wpisanie instrukcji dla bloków wejścia/wyjścia i przetwarzania danych. Jak zapewne zauważyłeś bloki Start i Koniec nie wymagają żadnego opisu. W celu wprowadzania tekstu należy zaznaczyć odpowiedni blok a następnie w oknie narzędzia(patrz rozdział 2.2.4) wprowadzić interesujące nas polecenia.
Naszym zadaniem (tak jak zostało powiedziane na początku rozdziału) będzie napisanie algorytmu obliczającego pole kwadratu. W tym celu wprowadź następujące instrukcje:

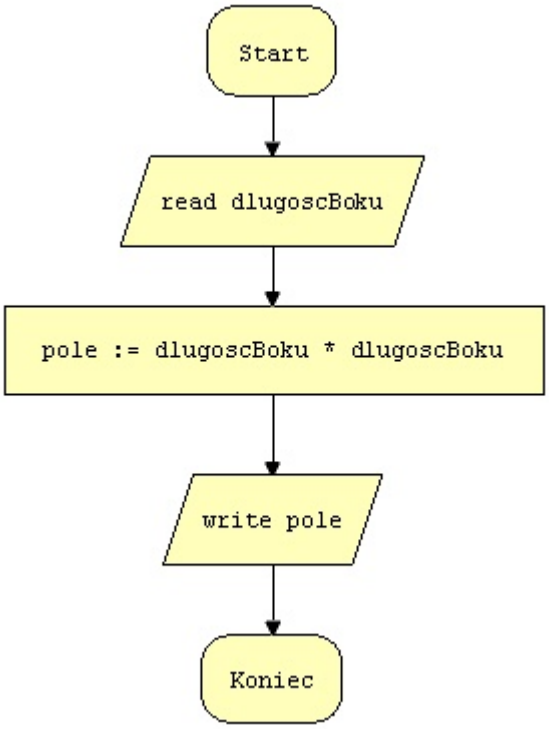
Dla pierwszego od góry bloku wejścia/wyjścia: `read dlugoscBoku`
Dla bloku przetwarzania danych: `pole := dlugoscBoku * dlugoscBoku`
Dla ostatniego bloku wejścia/wyjścia: `write pole`
Jeżeli wykonałeś wszystkie czynności prawidłowo Twój algorytm powinien wyglądać tak jak to zostało przedstawione na rysunku 4.1c.




Rysunek 4.1a




Rysunek 4.1b



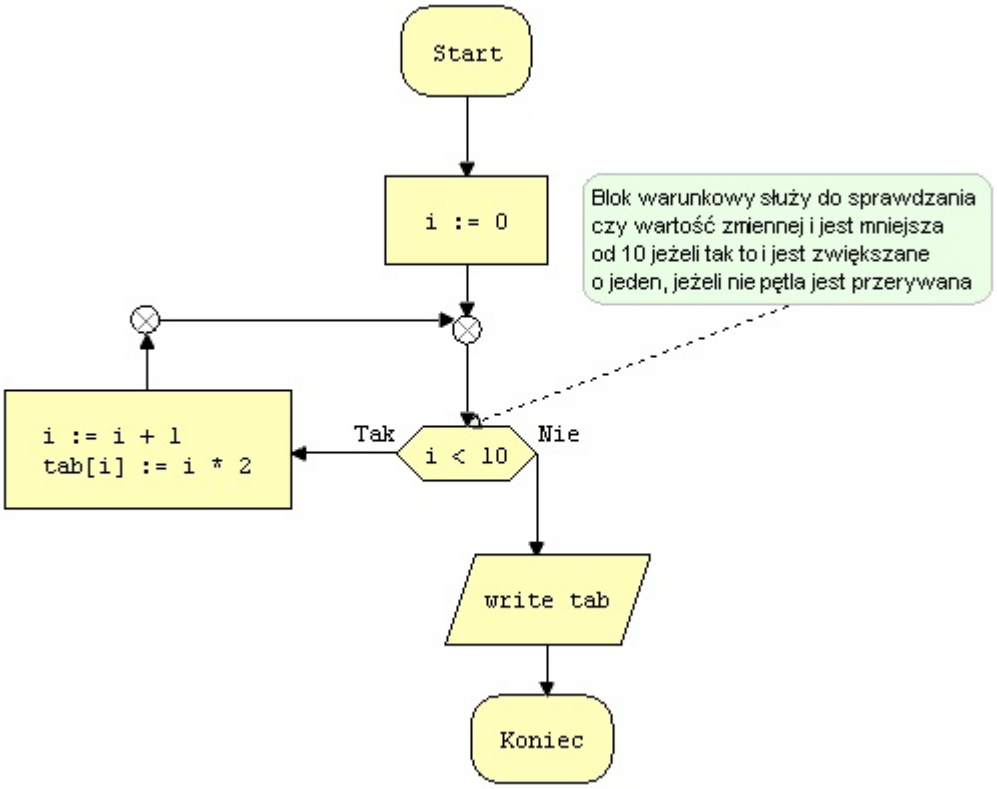

Rysunek 4.1c

Kolejny przykład będzie przedstawiał algorytm, którego rolą będzie wypełnienie tablicy 10 elementowej liczbami parzystymi. W związku z faktem że algorytm będzie zawierał pętlę zostanie wykorzystany blok warunkowy (Skrót klawiszowy Ctrl + 6) oraz dwa węzły pomocnicze (Skrót klawiszowy Ctrl + 8). Dodatkowo w celu objaśnienia działania pętli zostanie dodany blok opisowy (Skrót klawiszowy Ctrl + 9). Bloki opisowe nie są wymagane do poprawnego działania algorytmu ale ułatwiają późniejsze jego zrozumienie.

Projektowanie algorytmu rozpocznijmy od stworzenia nowego schematu oraz wstawienia wymienionych poniżej bloków:

- Blok Start
- Dwa bloki przetwarzania danych
- Dwa węzły pomocnicze
- Blok warunkowy
- Blok wejścia/wyjścia
- Blok Koniec

Kolejny etap to połączenie bloków i wprowadzenie instrukcji. Szczegóły można zobaczyć na rysunku 4.1d.




Rysunek 4.1d

My skupimy uwagę tylko na nowych elementach algorytmu czyli bloku warunkowym, węzłach pomocniczych i oplu opisowym.

Blok warunkowy tak jak to widać na powyższym rysunku może mieć dwa połączenia jednocześnie co oznacza że mogą istnieć dwa bloki docelowe. W tym wypadku kolejność wykonywania poszczególnych bloków zależy od tego czy warunek jest spełniony czy nie. Jeżeli warunek jest spełniony zawsze wykonuje się blok na który wskazuje połączenie oznaczone słowem „Tak", natomiast w przypadku kiedy warunek nie jest spełniony zawsze wykonuje się blok na który wskazuje połączenie oznaczone słowem „Nie". Warto dodać że połączenie z prawej strony bloku wykonuje się tak jak zwykłe połączenie z tą tylko różnicą że zamiast przyciśnięcia klawisza Ctrl(w trakcie wskazywania bloku docelowego) należy nacisnąć klawisz Shift. Dodatkowo okno Narzędzia udostępnia opcje, która umożliwia przełączanie oznaczeń połączeń Tak/Nie na przeciwne.

Węzły pomocnicze przydatne są w dwóch przypadkach:

- jeżeli kilka połączeń zbiega się w jednym miejscu(na rysunku 4.1d jest to węzeł pomiędzy blokiem przetwarzania danych a blokiem warunkowym)
- jeżeli chcemy stworzyć kąty proste pomiędzy połączeniami(na rysunku 4.1d węzeł ten znajduje się pomiędzy blokiem przetwarzania i węzłem pomocniczym)

Warto dodać że węzły pomocnicze można ukrywać za pomocą opcji Widok->Pokaż/ukryj węzły.

# 5. Analiza algorytmów

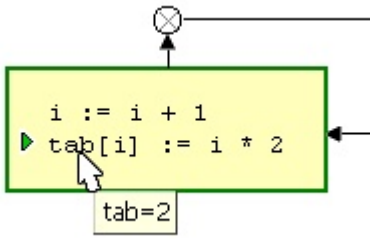
W tym rozdziale postaram się opisać różne sposoby analizowania algorytmu za pomocą omawianego programu. W tym celu będziemy posługiwać się algorytmem zaprojektowanym w poprzednim rozdziale(Rysunek 4.1d).

Analiza opiera się głównie na sprawdzaniu wartości zmiennych i tablic w trakcie wykonywania kolejnych bloków algorytmu. Pierwszym krokiem jaki należy wykonać jest przejście w tryb analizy. W tym celu włączmy okno analizy(Opcja Widok->Analiza programu skrót klawiszowy Ctrl + R) i naciśnijmy przycisk oznaczony symbolem  (Skrót klawiszowy F4 lub Ctrl + F9). Spowoduje to uruchomienie algorytmu w trybie krok po kroku. Zwróćmy uwagę że w prawym dolnym rogu głównego okna zmieni się informacja o trybie pracy z trybu projektowania na tryb analizy. Przejście z powrotem w tryb projektowania odbywa się w sposób automatyczny w chwili kiedy zaznaczymy, przesuniemy lub zmienimy instrukcję w dowolnym bloku.

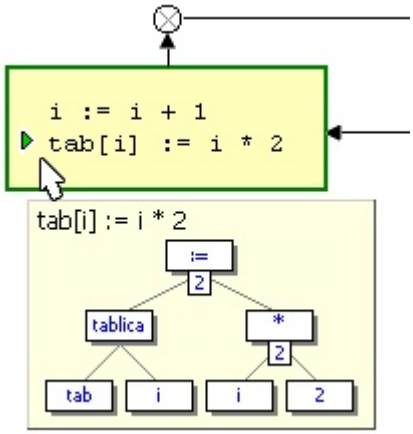
Zmiana trybu pracy na tryb analizy spowoduje wyświetlenie w oknie Analiza algorytmu wszystkich zmiennych używanych w algorytmie. W naszym przypadku będą to dwie zmienne o nazwach *i* oraz *tab*. Po prawej stronie każdej zmiennej można zobaczyć jej wartość w naszym przypadku na razie będą to znaki zapytania(równoważne ze słowem kluczowym *nil* lub wartością *NULL*), ponieważ zmienne nie mają jeszcze żadnych wartości. Spróbujmy teraz przejść do kolejnej instrukcji naciskając przycisk oznaczony symbolem  (Skrót klawiszowy F8). W tym momencie algorytm przejdzie i wykona kolejną instrukcję, w tym przypadku również przejdzie do następnego bloku. Jednocześnie kolorem zielonym zostanie wyróżnione obramowanie bloku(kolor zielony oznacza aktualnie przetwarzany blok).

Przejedzmy teraz do następnej instrukcji wciskając ponownie przycisk oznaczony symbolem  . Jak widać została wykonana instrukcja `i:=0`, można to zaobserwować na liście zmiennych(okno Analiza algorytmu). Wartość zmiennej *i* jest równa 0. Spróbuj teraz przechodzić do kolejnych instrukcji aż do momentu zakończenia algorytmu. Równocześnie obserwuj jak zmieniają się wartości zmiennych na liście zmiennych.

Dodatkowym ułatwieniem z jakiego możemy korzystać w trybie analizy to podglądanie wartości zmiennych bezpośrednio w blokach. W tym celu wystarczy przesunąć kursor nad nazwę zmiennej w bloku(Rysunek 5a). Jeżeli chcemy podglądnąć wszystkie mikroinstrukcję oraz ich kolejność wykonywania wystarczy przesunąć kursor myszki w miejsce gdzie znajduje się zielony trójkąt przed instrukcją(Rysunek 5b).




Rysunek 5a




Rysunek 5b

Drzewo mikroinstrukcji dostępne jest także na zakładce *Drzewo* znajdującej się na oknie *Kod źródłowy* (patrz rozdział 2.2.3 podpunkt c).

# Dodatki

## *Dodatek A – Opis wszystkich słów kluczowych*

| Słowo kluczowe | Opis | Przykład |
|---|---|---|
| `Dim` | Umożliwia deklarowanie zmiennych oraz tablic. Jeżeli chcemy zadeklarować tablicę musimy podać jej rozmiar. | `Dim zmienna`<br>`Dim a,b,c`<br>`Dim tablica[10]` |
| `Set` | Umożliwia przypisywać wartości dla całej tablicy. | `Set tab := (1,2,3)`<br>`Set tab := ((m11,m21),(m21,m22))`<br>`Set tab := ('ab','d')`<br>`Set tab := ('a',4,True)`<br>`Set tab (1,(2,3,4)` |
| `Read` | Umożliwia wprowadzanie danych z klawiatury poprzez konsolę wejścia/wyjścia | `Read zmienna`<br>`Read tablica`<br>`Read tab[5]`<br>`Read a,b,tab[3,8]` |
| `Write` | Umożliwia wyświetlanie wartości zmiennych oraz tablic poprzez konsolę wejścia/wyjścia | `Write zmienna`<br>`Write tablica`<br>`Write tab[5]`<br>`Write a,b,tab[3,8]`<br>`Write 'Zwykły tekst'` |

## *Dodatek B – Kolejność wykonywania operatorów*

Kolejność wykonywania operatora jest związana z jego priorytetem. W pierwszej kolejności wykonywane są operatory o priorytecie 1 natomiast w ostatniej kolejności z priorytetem 9. Operatory o takim samym priorytecie wykonywane są w kolejności ich występowania od lewej strony. Poniżej znajduje się tabela przedstawiająca priorytety operatorów:

| Operator | Priorytet |
|---|---|
| `( ) [ ]` | 1 |
| `! not ~ ++ --` | 2 |
| `* / div % mod` | 3 |
| `+ -` | 4 |
| `<< shl >> shr` | 5 |
| `< <= > >=` | 6 |
| `== != <>` | 7 |
| `& \|` | 8 |
| `&& \|\| and or` | 9 |
| `=` | 10 |
| `,` | 11 |

## *Dodatek C – Opis wszystkich procedur i funkcji*

### Funkcje matematyczne

| Funkcja | Opis | Przykład |
|---|---|---|
| `Random` | Zwraca losową liczbę z podanego zakresu | `a := Random(zakres)` |
| `Sqrt` | Oblicza pierwiastek | `a := Sqrt( 9 )` |
| `Sqr` | Oblicza potęga drugiego stopnia(kwadrat) | `a := Sqr( 3 )` |
| `Cos` | Oblicza kosinus z podanego kąta | `a := Cos( kat )` |
| `Sin` | Oblicza sinus z podanego kąta | `a := Sin( kat )` |
| `Tan` | Oblicza tangens z podanego kąta | `a := Tan( zakres )` |
| `Cotan` | Oblicza kotangens z podanego | `a := Cotan( zakres )` |
| `Cosh` | Oblicza kosinus hiperboliczny z podanego | `a := Cosh( kat )` |
| `Sinh` | Oblicza sinus hiperboliczny z podanego kąta | `a := Sinh( kat )` |
| `Tanh` | Oblicza tangens hiperboliczny z podanego | `a := Tanh( zakres )` |
| `Coth` | Oblicza kotangens hiperboliczny z podanego | `a := Coth( zakres )` |

Inne funkcje matematyczne z jednym argumentem:

| | | | | | |
|---|---|---|---|---|---|
| `ArcCos` | `ArcCosh` | `ArcCtg` | `ArcCtgH` | `ArcCsc` | `ArcCscH` |
| `ArcSec` | `ArcSecH` | `ArcSin` | `ArcSinh` | `ArcTan` | `ArcTanh` |
| `Cosecant` | `Cot` | `Csc` | `CscH` | `Sec` | `Secant` |
| `SecH` | | | | | |

Inne funkcje matematyczne z dwoma argumentami:

| | |
|---|---|
| `ArcTan2` | `Hypot` |

### Funkcje tekstowe

| Funkcja | Opis | Przykład |
|---|---|---|
| `Length` | Zwraca długość zmiennej tekstowej lub tekstu | `a := Length( 'abc' )` |
| `Copy` | Kopiuje tekst od pozycji podanej jako indeks oraz odpowiedniej długości | `tekst:=Copy(Str,indeks,dlugosc)` |

| | | |
|---|---|---|
| | | `a := Copy('abcde', 2, 3)`<br>`Wykik: a = 'bcd'` |
| `Pos` | Podaje pozycje podciągu w tekście | `a := Pos(podciag, tekst)`<br><br>`a := Pos('cd','abcde')`<br>`Wynik: a = 3` |
| `Insert` | Wstawia podciąg(tekst) do innego tekstu | `a:=Insert(podciag,tekst,indeks)`<br><br>`a := Insert('abc', 'nowy', 2)`<br>`Wynik: a = 'anowybc'` |
| `Delete` | Usuwa wybrany fragment tekstu | `a:=Delete(podciag,indeks,dlugosc)`<br><br>`a := Delete('abcde', 3, 2)`<br>`Wynik: a = 'abe'` |
| `Str` | Zamienia liczbę całkowitą lub rzeczywistą na tekst | `tekst := Str(liczba)`<br><br>`tekst := Str(56)`<br>`Wynik tekst = '56'` |
| `Val` | Zamienia liczbę w postaci tekstu na zwykłą liczbę | `liczba := Val(tekst)`<br><br>`liczba := Val('56')`<br>`Wynik: liczba = 56` |